Integrating Probability with Time and Shared-Variable Concurrency *

Huibiao Zhu
Software Engineering Institute
East China Normal University
hbzhu@sei.ecnu.edu.cn

Shengchao Qin
Department of Computer Science
University of Durham
shengchao.qin@durham.ac.uk

Jifeng He
Software Engineering Institute
East China Normal University
jifeng@sei.ecnu.edu.cn

Jonathan P. Bowen
Centre for Applied Formal Methods
London South Bank University
bowenjp@lsbu.ac.uk

Abstract

Complex software systems typically involve features like time, concurrency and probability, where probabilistic computations play an increasing role. It is challenging to formalize languages comprising all these features. In this paper, we integrate probability, time and concurrency in one single model, where the concurrency feature is modelled using shared-variable based communication. The probability feature is represented by a probabilistic nondeterministic choice, probabilistic guarded choice and a probabilistic version of parallel composition. We formalize an operational semantics for such an integration. Based on this model we define a bisimulation relation, from which an observational equivalence between probabilistic programs is investigated and a collection of algebraic laws are explored. We also implement a prototype of the operational semantics to animate the execution of probabilistic programs.

1 Introduction

As probabilistic computations play an increasing role in solving various problems [21], various proposals on probabilistic languages have been reported [5, 6, 8, 14, 13, 15, 18, 19, 20]. Complex software systems typically involve important features like real-time, probability and shared-variable concurrency. Therefore, system designers would expect a formal model that incorporates all these features to be available for them to use. However, to the best of our knowledge, no one has integrated all these features in one model. In this paper we tackle this challenging problem by proposing a formal model for a language equipped with probability, time and shared-variable concurrency. Our model is meant to facilitate the specification of complex software systems.

The shared-variable mechanism is typically used for communications among components running in parallel. Although shared-variable concurrency can be seen in many languages (e.g. the Java programming language, the Verilog hardware description language), it proves to be challenging to formalize it [11, 23, 24], not to mention other orthogonal features like probability and time. In this paper we successfully tackle this challenge by integrating time, probability and shared-variable concurrency in one model. The probability feature is reflected by the probabilistic nondeterministic choice, probabilistic guarded choice and the probabilistic scheduling of actions from different concurrent components in a program.

As advocated in Hoare and He's Unifying Theories of Programming (*UTP*) [12], three different styles of mathematical representations are normally used: operational, denotational, and algebraic ones, among which the operational style is the most intuitive one. In order to elaborate more on the intuition behind the proposed language and to formally define the behaviour of its programs, we start with the operational semantics in this paper. Upon the operational model, we define a bisimulation relation, from which a collection of algebraic laws are derived. The completeness of the algebraic laws and the Galois connection between the operational and algebraic theories are beyond the scope of this paper and would be addressed in future work.

Much related work has investigated the semantics for probabilistic processes. Morgan and his colleagues explored the abstraction and refinement for probabilistic processes using the weakest precondition (wp) approach [13, 14, 15]. Hartog and her colleagues have studied the equivalence between operational and denotational semantics for

*Partially supported by National Basic Research Program of China (No. 2002CB312001 and No. 2005CB321904) and the 211 project of The Ministry of Education of China.

Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop SEW-30 (SEW'06)
0-7695-2624-1/06 $20.00 © 2006 IEEE

a variety of probabilistic processes [5, 6, 7, 8] using Banach Space approach [4]. Núñez extended Henessey's "testing semantics" for a variety of probabilistic processes [18, 19, 20]. As an extension of the guarded command language, a simple probabilistic guarded command language was formalized in [10] under the *UTP* framework. A set of algebraic laws were then explored based on the denotational model.

The rest of this paper is organized as follows. Section 2 presents our language equipped with probability, time and shared-variable concurrency. Section 3 is devoted to the operational semantics. A bisimulation relation is then defined in Section 4 and used for the basis of a set of algebraic laws in Section 5. Section 6 gives a prototype animation of the operational semantics and Section 7 concludes the paper.

2 Probabilistic Language *PTSC*

In this paper we propose a probabilistic language *PTSC* (Probability, Time and Shared-variable Concurrency), which involves the integration of probability, time and shared-variable concurrency. Apart from the concurrency feature that exists in many conventional languages, probability and time are the other two main features of our language. The synchronization of different parallel components is based on time controls. Overall, our language consists of the following main features:

(1) Probabilistic behaviour: This can be represented by probabilistic nondeterminism, probabilistic guarded choice or probabilistic parallel composition.

(2) Timed behaviour: This can be reflected by event guard $@\, b$ (wait until b is satisfied) and time-delay command.

(3) Shared-variable concurrency: The concurrency model employs a shared-variable based communication mechanism.

The *PTSC* language has the following syntactical elements:

$$\begin{aligned} P ::= \ & \mathbf{Skip} \mid x := e \mid \mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ P \\ & \mid \mathbf{while}\ b\ \mathbf{do}\ P \mid @b\ P \mid \#n\ P \mid P\ ;\ P \\ & \mid P \sqcap P \mid P \sqcap_p P \mid P \parallel_p P \end{aligned}$$

Note that:

(1) $x := e$ is the atomic assignment. $\mathbf{Skip}$ behaves the same as $x := x$.

(2) Regarding $@b\ P$, when the Boolean condition b is satisfied, process P can have the chance to be scheduled. As we consider models for closed systems, the program $@b\ P$ can only let time advance when the Boolean condition b is not met. For $\#n\ P$, after n time units elapse, process P can be scheduled.

(3) Similar to a conventional programming language, $\mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ P$ stands for the conditional, whereas $\mathbf{while}\ b\ \mathbf{do}\ P$ stands for the iteration.

(4) The mechanism for parallel composition $P \parallel_p Q$ is a shared-variable interleaving model with probability feature. If process P can perform an atomic action, $P \parallel_p Q$ has conditional probability p to do that atomic action. On the other hand, if process Q can perform an atomic action, $P \parallel_p Q$ has conditional probability $1-p$ to perform that action.

(5) $\sqcap$ stands for the nondeterministic choice, where $\sqcap_p$ stands for the probabilistic nondeterministic choice. $P \sqcap_p Q$ indicates that the probability for $P \sqcap_p Q$ to behave as P is p, where the probability for $P \sqcap_p Q$ to behave as Q is $1-p$.

In order to facilitate algebraic reasoning, we enrich our language with a *guarded choice* in our language. As our parallel composition has probability feature, the guarded choice also shares this feature. Guarded choice is classified into five types:

(1) $[\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$

(2) $[\!]_{i\in I}\{@b_i\ P_i\}$

(3) $[\!]\{\#1\ R\}$

(4) $[\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$
$[\!][\!]_{k\in K}\{@b_k\ Q_k\}$

(5) $[\!]_{i\in I}\{@b_i\ P_i\}[\!]\{\#1\ R\}$

Regarding $[\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$ in the guarded choice type (1) and (4), it should satisfy the following healthiness conditions:

(a) $\forall i \bullet (\bigvee_{j\in J_i} b_{ij} = true)$ and
$(\forall j_1, j_2 \bullet (j_1 \neq j_2) \Rightarrow ((b_{ij_1} \wedge b_{ij_2}) = false))$

(b) $+_{i\in I}\, p_i = 1$

The first type is composed of a set of assignment-guarded components. The condition (a) indicates that for any $i \in I$, the Boolean conditions b_{ij} from "$\mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})$" are complete and disjoint. Therefore, there will be exactly one component $b_{ij}\&(x_{ij} := e_{ij})\ P_{ij}$ selected among all $j \in J_i$. Furthermore, for any $i \in I$, the possibility for a component $(x_{ij} := e_{ij})\ P_{ij}$ (where b_{ij} is met) to be scheduled is p_i and it should satisfy the second healthiness condition.

The second type is composed of a set of event-guarded components. If one guard is satisfied, the subsequent behaviour for the whole process will be followed by its subsequent behaviour of the satisfied component.

The third type is composed of one time delay component. Initially, it cannot do anything except letting time advance one unit.

The fourth type is the guarded choice composition of the first and second type of guarded choice. If there exists one b_k ($k \in K$) being satisfied currently, then the event $@\, b_k$ is

fired and the subsequent behaviour is Q_k. If there is no satisfied b_k, the behaviour of the fourth type of guarded choice is the same as that of the first type.

The fifth type is the compound of the second and third type of guarded choice. Currently, if there exists i ($i \in I$) such that b_i is satisfied, then the subsequent behaviour of the whole guarded choice is as P_i. On the other hand, if there is no i ($i \in I$) such that b_i is satisfied currently, then the whole guarded choice cannot do anything initially except letting time advance one unit. The subsequent behaviour is the same as the behaviour of R.

As the first type of guarded choice does not have time advancing behavior, there is no type of guarded choice composing of the first and third type of guarded choice.

3 Operational Semantics

The operational semantics of a language models the behaviour of a program in terms of execution steps, which are represented by transition relations. In our operational model, the transitions are expressed in the form of Plotkin's Structural Operational Semantics (SOS) [22]:

$$\langle P,\ \sigma\rangle \ \xrightarrow{\beta}\ \langle P',\ \sigma'\rangle$$

where, P stands for the program text that remains to be executed. σ is the current state of the program.

The transitions can be classified into four kinds:

(1) The first kind of transitions models an atomic action with certain probability. In this paper, we consider assignment as an atomic action.

$$\langle P,\ \sigma\rangle \ \xrightarrow{c}_p\ \langle P',\ \sigma'\rangle$$

where, p stands for the probability for program P to perform the execution.

(2) The second type models the transition of a time delay. Time advances in unit steps.

$$\langle P,\ \sigma\rangle \ \xrightarrow{1}\ \langle P',\ \sigma'\rangle$$

(3) The third type models the selection of the two components for non-deterministic choice. It can be expressed as:

$$\langle P,\ \sigma\rangle \ \xrightarrow{\tau}\ \langle P',\ \sigma\rangle$$

(4) The fourth type models the triggered case of event @ b:

$$\langle P,\ \sigma\rangle \ \xrightarrow{v}\ \langle P',\ \sigma\rangle$$

In what follows, we shall present the operational rules for sequential programs, probabilistic guarded choice, and concurrent programs.

3.1 Sequential Process

A sequential program comprising a single assignment performs an atomic action with probability 1. It cannot perform any other types of transitions.

$$\langle x := e,\ \sigma\rangle \ \xrightarrow{c}_1\ \langle \varepsilon,\ \sigma[e/x]\rangle$$

where, ε stands for the empty process.

For the conditional statement $\mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ Q$, the control flow will be passed to P with probability 1 if b is satisfied, otherwise it will be passed to Q with probability 1.

$$\langle \mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ Q\ ,\ \sigma\rangle \ \xrightarrow{c}_1\ \langle P,\ \sigma\rangle, \quad \text{if } b(\sigma)$$
$$\langle \mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ Q\ ,\ \sigma\rangle \ \xrightarrow{c}_1\ \langle Q,\ \sigma\rangle, \quad \text{if } \neg b(\sigma)$$

Here $b(\sigma)$ returns the value of b in the state σ.

The transitions for iteration are similar to conditional.

$$\langle \mathbf{while}\ b\ \mathbf{do}\ P,\ \sigma\rangle \xrightarrow{c}_1 \langle P\,;\ \mathbf{while}\ b\ \mathbf{do}\ P,\ \sigma\rangle, \quad \text{if } b(\sigma)$$
$$\langle \mathbf{while}\ b\ \mathbf{do}\ P,\ \sigma\rangle \xrightarrow{c}_1 \langle \varepsilon,\ \sigma\rangle, \quad \text{if } \neg b(\sigma)$$

Time delay can advance time in unit steps. It cannot do any other types of transitions.

$$\langle \#n,\ \sigma\rangle \ \xrightarrow{1}\ \langle \#(n-1),\ \sigma\rangle, \quad \text{where } n > 1.$$
$$\langle \#1,\ \sigma\rangle \ \xrightarrow{1}\ \langle \varepsilon,\ \sigma\rangle$$

The event $@b$ in $@b\ P$ is satisfied if Boolean condition b is currently satisfied, otherwise it will let time advance one unit. We use "$\xrightarrow{v}$" to model the event-triggered transition instead of "$\xrightarrow{c}$'.

$$\langle @b\ P,\ \sigma\rangle \ \xrightarrow{v}\ \langle P,\ \sigma\rangle, \quad \text{if } b(\sigma)$$
$$\langle @b\ P,\ \sigma\rangle \ \xrightarrow{1}\ \langle @b\ P,\ \sigma\rangle, \quad \text{if } \neg b(\sigma)$$

The selection of process P or Q from $P \sqcap Q$ is nondeterministic.

$$\langle P \sqcap Q,\ \sigma\rangle \ \xrightarrow{\tau}\ \langle P,\ \sigma\rangle$$
$$\langle P \sqcap Q,\ \sigma\rangle \ \xrightarrow{\tau}\ \langle Q,\ \sigma\rangle$$

For the probabilistic nondeterministic choice $P \sqcap_p Q$, the probability of the selection of P is p and the probability of the selection of Q is $1-p$.

$$\langle P \sqcap_p Q,\ \sigma\rangle \ \xrightarrow{c}_p\ \langle P,\ \sigma\rangle$$
$$\langle P \sqcap_p Q,\ \sigma\rangle \ \xrightarrow{c}_{1-p}\ \langle Q,\ \sigma\rangle$$

The process $P\ ;\ Q$ will behave like P initially. After P terminates, Q will be executed.

$$\text{if } \langle P,\ \sigma\rangle \xrightarrow{\beta} \langle \varepsilon,\ \sigma'\rangle, \quad \text{then } \langle P;Q,\ \sigma\rangle \xrightarrow{\beta} \langle Q,\ \sigma'\rangle$$
$$\text{if } \langle P,\ \sigma\rangle \xrightarrow{\beta} \langle P',\ \sigma'\rangle, \text{ then } \langle P;Q,\ \sigma\rangle \xrightarrow{\beta} \langle P';Q,\ \sigma'\rangle$$

where $\xrightarrow{\beta}$ can be $\xrightarrow{\tau}$, $\xrightarrow{v}$, $\xrightarrow{c}_p$ and $\xrightarrow{1}$

3.2 Probabilistic Guarded Choice

In order to investigate algebraic properties for parallel composition, we enrich the language with guarded choice. A guarded choice construct may perform transitions depicted in the following five cases.

(1) Let $P = [\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$.

$\langle P,\ \sigma\rangle \xrightarrow{c}_{p_i} \langle P_{ij},\ \sigma[e_{ij}/x_{ij}]\rangle, \quad$ if $b_{ij}(\sigma)$

For any $i \in I$, there is only one $j \in J_i$ such that $b_{ij}(\sigma) = true$. This indicates that program P can execute assignment $x_{ij} := e_{ij}$ with probability p_i when the corresponding condition $b_{ij}(\sigma) = true$.

(2) Let $P = [\!]_{i\in I}\{@b_i\, P_i\}$.

$\langle P,\ \sigma\rangle \xrightarrow{v} \langle P_i,\ \sigma\rangle, \quad$ if $b_i(\sigma)$

$\langle P,\ \sigma\rangle \xrightarrow{1} \langle P,\ \sigma\rangle, \quad$ if $\bigwedge_{i\in I} b_i(\sigma) = false$

If there exists i ($i \in I$) such that $b_i(\sigma)$ is satisfied, the event $@b_i$ is enabled, thus P_i is followed, as depicted in the first rule. If $\forall i \bullet b_i(\sigma) = false$, no events are enabled, only time can advance, as indicated in the second rule.

(3) Let $P = [\!]\{\#1\, R\}$.

$\langle P,\ \sigma\rangle \xrightarrow{1} \langle R,\ \sigma\rangle$

Process P can only do time advancing transition because it only contains time-delay component. The subsequent behaviour after one time unit elapses is just the behaviour of process R.

(4) Let $P = [\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J}(b_{ij}\&(x_{ij} := e_{ij})\, P_{ij})\}$
$[\!][\!]_{k\in K}\{@c_k\, Q_k\}$

$\langle P,\ \sigma\rangle \xrightarrow{v} \langle Q_k,\ \sigma\rangle, \quad$ if $c_k(\sigma) = true$

$\langle P,\ \sigma\rangle \xrightarrow{c}_{p_i} \langle P_{ij},\ \sigma[e_{ij}/x_{ij}]\rangle,$
if $b_{ij}(\sigma) \wedge (\forall k \bullet c_k(\sigma) = false)$.

For any $k \in K$, if $c_k(\sigma) = true$, then the event "$@c_k$" is fired. The first transition reflects this fact. For any $i \in I$, if $b_{ij}(\sigma) = true$ and $\forall k \bullet c_k(\sigma) = false$, then process P can perform the corresponding assignment "$x_{ij} := e_{ij}$". Now consider the special case "$c_k(\sigma) = true$" and $b_{ij}(\sigma) = true$, we only allow event "$@c_k$" to be fired and do not allow process P to perform assignment $x_{ij} := e_{ij}$. This fact is shown in the first transition and reflected by the additional condition "$\forall c_k \bullet c_k(\sigma) = false$" in the second transition.

(5) Let $P = [\!]_{i\in I}\{@b_i\, P_i\}[\!]\{\#1\, R\}$.

$\langle P,\ \sigma\rangle \xrightarrow{v} \langle P_i,\ \sigma\rangle$ if $b_i(\sigma)$

$\langle P,\ \sigma\rangle \xrightarrow{1} \langle R,\ \sigma\rangle \quad$ if $\bigwedge_{i\in I} b_i(\sigma) = false$

The first transition indicates that event "$@b_i$" is fired if $b_i(\sigma) = true$. However, if all $b_i(\sigma)$ are evaluated to $false$, then only time-advance branch can be selected.

3.3 Parallel Process

For brevity of presentation, we first define the following function to represent intermediate processes:

$$\mathbf{par}(P, Q, p_1) =_{df} \begin{cases} P \parallel_{p_1} Q & \text{if } P \neq \varepsilon \text{ and } Q \neq \varepsilon \\ P & \text{if } P \neq \varepsilon \text{ and } Q = \varepsilon \\ Q & \text{if } P = \varepsilon \text{ and } Q \neq \varepsilon \\ \varepsilon & \text{if } P = \varepsilon \text{ and } Q = \varepsilon \end{cases}$$

This intermediate format can reduce the number of transitions for parallel composition by representing several cases in one single rule.

Now we define two functions:

$stable(\langle P, \sigma\rangle) =_{df} \neg(\ \langle P, \sigma\rangle \xrightarrow{\tau}\ )$ and

$stableE(\langle P, \sigma\rangle) =_{df} \neg(\ \langle P, \sigma\rangle \xrightarrow{v}\ )$

The notation $stable(\langle P, \sigma\rangle)$ indicates that process P cannot perform the transition representing nondeterministic choice under state σ, while $stableE(\langle P, \sigma\rangle)$ indicates that process P cannot perform event-triggered transitions under state σ.

A probabilistic parallel composition may perform transitions of the following forms:

(1) (a) If $\langle P,\ \sigma\rangle \xrightarrow{\tau} \langle P',\ \sigma\rangle$ and $stable(\langle Q, \sigma\rangle)$,
then $\langle P \parallel_{p_1} Q,\ \sigma\rangle \xrightarrow{\tau} \langle \mathbf{par}(P', Q, p_1),\ \sigma\rangle$.
$\langle Q \parallel_{p_1} P,\ \sigma\rangle \xrightarrow{\tau} \langle \mathbf{par}(Q, P', p_1),\ \sigma\rangle$.

(b) If $\langle P,\ \sigma\rangle \xrightarrow{\tau} \langle P',\ \sigma,\rangle$ and
$\langle Q,\ \sigma\rangle \xrightarrow{\tau} \langle Q',\ \sigma\rangle$,
then $\langle P \parallel_{p_1} Q,\ \sigma\rangle \xrightarrow{\tau} \langle \mathbf{par}(P', Q', p_1),\ \sigma\rangle$

(2) (a) If $\langle P,\ \sigma\rangle \xrightarrow{v} \langle P',\ \sigma\rangle$ and $stable(\langle Q, \sigma\rangle)$
and $stableE(\langle Q, \sigma\rangle)$,
then $\langle P \parallel_{p_1} Q,\ \sigma\rangle \xrightarrow{v} \langle \mathbf{par}(P', Q, p_1),\ \sigma\rangle$.
$\langle Q \parallel_{p_1} P,\ \sigma\rangle \xrightarrow{v} \langle \mathbf{par}(Q, P', p_1),\ \sigma\rangle$.

(b) If $\langle P,\ \sigma\rangle \xrightarrow{v} \langle P',\ \sigma\rangle$ and
$\langle Q,\ \sigma\rangle \xrightarrow{v} \langle Q',\ \sigma\rangle$,
then $\langle P \parallel_{p_1} Q,\ \sigma\rangle \xrightarrow{v} \langle \mathbf{par}(P', Q', p_1),\ \sigma\rangle$

(3) If $\langle P,\ \sigma\rangle \xrightarrow{c}_{p_2} \langle P',\ \sigma'\rangle$ and
$stable(\langle x, \sigma\rangle)$ and $stableE(\langle x, \sigma\rangle)$ $(x = P,\ Q)$,
then $\langle P \parallel_{p_1} Q,\ \sigma\rangle \xrightarrow{c}_{p_1\times p_2} \langle \mathbf{par}(P', Q, p_1),\ \sigma'\rangle$
$\langle Q \parallel_{p_1} P,\ \sigma\rangle \xrightarrow{c}_{(1-p_1)\times p_2} \langle \mathbf{par}(Q, P', p_1),\ \sigma'\rangle$

(4) If $\langle P,\ \sigma\rangle \xrightarrow{1} \langle P',\ \sigma'\rangle$ and $\langle Q,\ \sigma\rangle \xrightarrow{1} \langle Q',\ \sigma'\rangle$
and
$stable(\langle x, \sigma\rangle)$ and $stableE(\langle x, \sigma\rangle)$ $(x = P,\ Q)$,
then $\langle P \parallel_{p_1} Q,\ \sigma\rangle \xrightarrow{1} \langle \mathbf{par}(P', Q', p_1),\ \sigma'\rangle$.

Transition (1)(a) stands for the case that one component makes nondeterministic choice and another component is

stable. The whole process also makes a nondeterministic choice under this case. However, if both components make nondeterministic choice, then the whole process can make nondeterministic choice and the subsequent behaviour is the parallel composition of the remaining components. Transition (1)(b) reflects this situation.

The second type stands for the event-fired case. The analysis is similar to the transitions of type (1). Transition (3) covers the case of performing an atomic action. If process P can perform an atomic action with probability p_2, then process $P \parallel_{p_1} Q$ and $Q \parallel_{p_1} P$ can also perform the same atomic action with probability $p_1 \times p_2$ and $(1-p_1)\times p_2$ respectively.

If both components can perform a time-advancing transition, then the whole parallel process can also let time advance. The aspect is reflected in transition (4).

4 Bisimulation

In operational semantics the behaviour of programs is represented in terms of execution steps. A computation is thus composed of a sequence of transitions. Two syntactically different programs may have the same observational behaviour. This means that we need to define program equivalence (conventionally denoted $\approx$) based on a reasonable abstraction. In considering the equivalence of the programs for our language, bisimulation is a useful approach. It will also be applied in exploring algebraic laws of our language.

In what follows we shall give several auxiliary definitions before we present the definition for bisimulation.

Definition 1 *We define the transition relation $\stackrel{id}{\Longrightarrow}_p$ as follows:*

$$\begin{aligned}
&\langle P,\ \sigma\rangle \stackrel{id}{\Longrightarrow}_p \langle P',\ \sigma\rangle \\
=_{df}\ &\textit{either } P' = P \textit{ and } p = 1 \\
&\quad \textit{or} \\
&\exists n, P_1, p_1, \ldots, P_n, p_n \bullet \\
&\quad \langle P,\ \sigma\rangle \stackrel{\beta_1}{\longrightarrow}_{p_1} \langle P_1,\ \sigma\rangle \ldots \stackrel{\beta_n}{\longrightarrow}_{p_n} \langle P_n,\ \sigma\rangle \\
&\quad \textit{and}\quad P_n = P' \quad \textit{and} \quad p = p_1 \times \ldots \times p_n
\end{aligned}$$

where $\stackrel{\beta_i}{\longrightarrow}_{p_i}$ can be of the forms $\stackrel{c}{\longrightarrow}_{p_i}$, $\stackrel{\tau}{\longrightarrow}$ and $\stackrel{v}{\longrightarrow}$. We assume $p_i = 1$ in the latter two cases.

Note that transition relation "$\stackrel{id}{\Longrightarrow}_p$" represents a sequence of transitions which keep the program state unchanged.

Definition 2 *We define the following two transition relations:*

(1)
$$\begin{aligned}
&\langle P,\ \sigma\rangle \stackrel{c}{\Longrightarrow}_p \langle P',\ \sigma'\rangle \\
=_{df}\ &\exists P_1 \bullet \langle P,\ \sigma\rangle \stackrel{id}{\Longrightarrow}_{p_1} \langle P_1,\ \sigma\rangle \textit{ and} \\
&\quad \langle P_1,\ \sigma\rangle \stackrel{c}{\longrightarrow}_{p_2} \langle P',\ \sigma'\rangle \textit{ and} \\
&\quad p = p_1 \times p_2
\end{aligned}$$

(2)
$$\begin{aligned}
&\langle P,\ \sigma\rangle \stackrel{1}{\Longrightarrow}_p \langle P',\ \sigma\rangle \\
=_{df}\ &\exists P_1 \bullet \langle P,\ \sigma\rangle \stackrel{id}{\Longrightarrow}_p \langle P_1,\ \sigma\rangle \textit{ and} \\
&\quad \langle P_1,\ \sigma\rangle \stackrel{1}{\longrightarrow} \langle P',\ \sigma\rangle
\end{aligned}$$

Note that the relation $\stackrel{c}{\Longrightarrow}_p$ is actually a composition of $\stackrel{id}{\Longrightarrow}_{p_1}$ and $\stackrel{c}{\longrightarrow}_{p_2}$ ($p = p_1 \times p_2$), while $\stackrel{1}{\Longrightarrow}_p$ is a composition of $\stackrel{id}{\Longrightarrow}_p$ and $\stackrel{1}{\longrightarrow}$. These two auxiliary relations will help us to present our bisimulation relation in a more abstract manner with respect to atomic actions and time-advancing.

Definition 3 *If a transition $\langle P,\ \sigma\rangle \stackrel{\beta}{\Longrightarrow}_{p_1} \langle P',\ \sigma'\rangle$ is duplicated n times, we denote it as*

$$\langle P,\ \sigma\rangle \stackrel{\beta}{\Longrightarrow}_{n,p_1} \langle P',\ \sigma'\rangle$$

where $\stackrel{\beta}{\Longrightarrow}_{p_1}$ can be of the form $\stackrel{c}{\Longrightarrow}_{p_1}$ or $\stackrel{1}{\Longrightarrow}_{p_1}$.

Definition 4 *A symmetric relation R is a bisimulation if and only if $\forall \langle P,\sigma\rangle R \langle Q,\sigma\rangle$*

(1) If $\langle P,\ \sigma\rangle \stackrel{x}{\longrightarrow} \langle P',\ \sigma\rangle$,
then $\exists Q' \bullet \langle Q,\ \sigma\rangle (\stackrel{\tau}{\longrightarrow} \ \vee\ \stackrel{v}{\longrightarrow})^* \langle Q',\ \sigma\rangle$ and $\langle P',\sigma\rangle R \langle Q',\sigma\rangle$.
where, $\stackrel{x}{\longrightarrow}$ can be of the transition type $\stackrel{\tau}{\longrightarrow}$ or $\stackrel{v}{\longrightarrow}$.

(2) If $\langle P,\ \sigma\rangle \stackrel{c}{\Longrightarrow}_{n_1,p_1} \langle P',\ \sigma'\rangle$,
(2-1) if $\sigma \neq \sigma'$, then $\exists Q', n_2, p_2 \bullet$
$\langle Q,\ \sigma\rangle \stackrel{c}{\Longrightarrow}_{n_2,p_2} \langle Q',\ \sigma'\rangle$ *and*
$\langle P',\sigma'\rangle R \langle Q',\sigma'\rangle$ *and*
$n_1 \times p_1 = n_2 \times p_2$.
(2-2) if $\sigma = \sigma'$, then
either
$\langle P',\sigma'\rangle R \langle Q,\sigma'\rangle$ *and* $n_1 \times p_1 = 1$
or $\exists Q', n_2, p_2 \bullet$
$\langle Q,\ \sigma\rangle \stackrel{c}{\Longrightarrow}_{n_2,p_2} \langle Q',\ \sigma'\rangle$ *and*
$\langle P',\sigma'\rangle R \langle Q',\sigma'\rangle$ *and*
$n_1 \times p_1 = n_2 \times p_2$.

(3) If $\langle P,\ \sigma\rangle \stackrel{1}{\Longrightarrow}_{n_1,p_1} \langle P',\ \sigma\rangle$,
then $\exists Q', n_2, p_2 \bullet$
$\langle Q,\ \sigma\rangle \stackrel{1}{\Longrightarrow}_{n_2,p_2} \langle Q',\ \sigma\rangle$ *and*
$\langle P',\sigma\rangle R \langle Q',\sigma\rangle$ *and*
$n_1 \times p_1 = n_2 \times p_2$.

Two configurations should have the same interface when studying their equivalence; i.e., their state parts should be the same. Our bisimulation relation is based on three different kinds of observations: a τ transition or a v-transition (triggered by an event), atomic action, and time advancing.

Note that (2-2) in the above definition models the case that if a process performs an atomic action without any contribution to the program state change, then its bisimilar process may or may not perform an atomic action with similar effect. This partly reflects the concept of *weak bisimulation* [16, 17].

Lemma 1 *If S_1 and S_2 are bisimulations, then the following relations are also bisimulations:*

(1) Id (2)$S_1 \circ S_2$ (3) $S_1 \cup S_2$

where, Id is the identity relation and $S_1 \circ S_2$ stands for the relational composition of S_1 and S_2.

Definition 5
(1) Two configurations $\langle P_1, \sigma\rangle$ and $\langle P_2, \sigma\rangle$ are bisimilar, written as $\langle P_1, \sigma\rangle \approx \langle P_2, \sigma\rangle$, if there exists a bisimulation relation R such that $\langle P_1, \sigma\rangle \; R \; \langle P_2, \sigma\rangle$.

(2) Two processes P and Q are bisimilar, denoted as $P \approx Q$, if for any state σ

$$\langle P,\ \sigma\rangle \approx \langle Q,\ \sigma\rangle$$

Lemma 2 *$\approx$ is an equivalence relation.*

Theorem 3 *$\approx$ is a congruence.*
Proof We can proceed the proof by structural induction. The detailed proof is left in Appendix B. □

This theorem indicates that bisimilar relation $\approx$ is preserved by all operators.

5 Algebraic Laws

Operational semantics can be used to deduce interesting properties of programs. In this section we investigate the algebraic laws of our timed language with probability and shared-variable concurrency.

For assignment, conditional, iteration, nondeterministic choice and sequential composition, our language enjoys similar algebraic properties as those reported in [9, 12]. In what follows, we shall only focus on novel algebraic properties with respect to time, probabilistic nondeterministic choice and parallel composition.

Two consecutive time delays can be combined into a single one, where the length of the delay is the sum of the original two lengths.

(delay-1) $\#n; \#m \;=\; \#(n+m)$

Probabilistic nondeterministic choice is idempotent.

(prob-1) $P \sqcap_p P = P$

However, it is not purely symmetric and associative. Its symmetry and associativity rely on the change of the associated probabilities:

(prob-2) $P \sqcap_{p_1} Q = Q \sqcap_{1-p_1} P$

(prob-3) $P \sqcap_p (Q \sqcap_q R) \;=\; (P \sqcap_x Q) \sqcap_y R$

where $x = p/(p+q-p\times q)$ and $y = p+q-p\times q$

Sequential composition also distributes through probabilistic nondeterministic choice.

(prob-4) $P; (Q \sqcap_{p_1} R) \;=\; (P;Q) \sqcap_{p_1} (P;R)$

(prob-5) $(P \sqcap_{p_1} Q); R \;=\; (P;R) \sqcap_{p_1} (Q;R)$

The proof for law (prob-3) is given in Appendix C. Other proofs are similar and omitted.

Probabilistic parallel composition is also not purely symmetric and associative. Its symmetry and associativity rely on the change of the associated probabilities as well.

(par-1) $P \parallel_p Q \;=\; Q \parallel_{1-p} P$

(par-2) $P \parallel_p (Q \parallel_q R) \;=\; (P \parallel_x Q) \parallel_y R$

where, $x = p/(p+q-p\times q)$ and $y = p+q-p\times q$

In what follows we give a collection of parallel expansion laws, which enable us to expand a probabilistic parallel composition to a guarded choice construct. As mentioned earlier, there exist five types of guarded choice. To take into account a parallel composition of two arbitrary guarded choices, we end up with fifteen different expansion laws.

The first five laws we shall discuss are with respect to parallel composition of assignment-guarded choice with any other choices. For brevity, we assume the first component is an assignment-guarded choice. In what follows, we shall list three laws, with the rest two left in Appendix A.

If the second component is also an assignment-guarded choice, the scheduling rule is that any assignment could be scheduled with the associated probability provided that its Boolean condition is satisfied. Suppose the assignment guard from the first component is scheduled, the subsequent behaviour is the parallel composition of the remaining process of the first component with the whole second component. Law (par-3-1) below depicts this case.

(par-3-1) Let

$P = \; [\!]_{i\in I}\{[p_i]\; \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\; P_{ij})\}$ and

$Q = \; [\!]_{k\in K}\{[q_k]\; \mathit{choice}_{l\in L_k}(b_{kl}\&(x_{kl} := e_{kl})\; P_{kl})\}$

Then

$$\begin{aligned} & P \parallel_r Q \\ = \; & [\!]_{i\in I}\{[r\times p_i]\; \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij}) \\ & \qquad\qquad \mathbf{par}(P_{ij}, Q, r)\} \\ & [\!][\!]_{k\in K}\{[(1-r)\times q_k]\; \mathit{choice}_{l\in L_k}(b_{kl}\&(x_{kl} := e_{kl}) \end{aligned}$$

$$\mathbf{par}(P, Q_{kl}, r)\}$$

If the second component is an event-guarded choice, the behaviour of the parallel composition can be described as the guarded choice of a set of assignment-guarded components and a set of event-guarded components. If an assignment guard (from the first component) is scheduled first, the subsequent behaviour is the parallel composition of the remaining part of the first component (P_{ij}) with the second component (Q); if an event guard is triggered, the subsequent behaviour is the parallel composition of the first component (P) with the remaining part of the second component (Q_k). This is presented in law (par-3-2).

(par-3-2) Let

$$P = [\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$$

and

$$Q = [\!]_{k\in K}\{@c_k\ Q_k\}$$

Then

$$\begin{aligned} & P \parallel_r Q \\ = {} & [\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ \mathbf{par}(P_{ij}, Q, r)\} \\ & [\!][\!]_{k\in K}\{@c_k\ \mathbf{par}(P, Q_k, r)\} \end{aligned}$$

If the second component is a time delay construct, then only assignment guards can be scheduled initially. For the whole parallel composition, the subsequent behaviour following the scheduled assignment guard is the parallel composition of the remaining part of the first component (P_{ij}) with the time delay component. The whole process does not have time delay component. This case is expressed in law (par-3-3).

(par-3-3) Let

$$P = [\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\} \quad \text{and}$$
$$Q = [\!]\{\#1\ R\}$$

Then

$$\begin{aligned} & P \parallel_r Q \\ = {} & [\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ \mathbf{par}(P_{ij}, Q, r))\} \end{aligned}$$

In what follows, we consider parallel compositions where the first component is an event-guarded choice, while the second component can be of any form. We shall present three laws (law (par-3-6) to (par-3-8)) with the rest listed in Appendix A.

If the second component is also an event-guarded choice, there are several scenarios. If one guard from the first component is triggered but no guards from the second component are triggered, the subsequent behaviour is the parallel composition of the remaining part of the first component with the second component. If two guards from both components are triggered simultaneously, the subsequent behaviour is the parallel composition of the remaining processes of both sides. This is illustrated in law (par-3-6).

(par-3-6) Let $P = [\!]_{i\in I}\{@b_i\ P_i\}$ and $Q = [\!]_{j\in J}\{@c_j\ Q_j\}$

Then

$$\begin{aligned} & P \parallel_r Q \\ = {} & [\!]_{i\in I}\{@(b_i \wedge \neg c)\ \mathbf{par}(P_i, Q, r)\} \\ & [\!][\!]_{j\in J}\{@(c_j \wedge \neg b)\ \mathbf{par}(P, Q_j, r)\} \\ & [\!][\!]_{i\in I\wedge j\in J}\{@(b_i \wedge c_j)\ \mathbf{par}(P_i, Q_j, r)\} \end{aligned}$$

where, $b = \vee_{i\in I}\, b_i$ and $c = \vee_{j\in J}\, c_j$

If the second component is a time delay construct, the whole process will wait for some events to be triggered. The whole process can also let time advance. Law (par-3-7) expresses this case.

(par-3-7) Let $P = [\!]_{i\in I}\{@b_i\ P_i\}$ and $Q = [\!]\{\#1\ R\}$

Then

$$P \parallel_r Q = [\!]_{i\in I}\{@b_i\ \mathbf{par}(P_i, Q, r)\}[\!]\{\#1\ \mathbf{par}(P, R, r)\}$$

If the second component is the guarded choice of a set of assignment-guarded components and a set of event-guarded components, any assignment guard can be scheduled. As both components have event-guard components, there are also three possibilities for the guards to be triggered, as discussed in (par-3-6). This case is described in law (par-3-8).

(par-3-8) Let $P = [\!]_{i\in I}\{@b_i\ P_i\}$ and

$$Q = [\!]_{j\in J}\{[q_j]\ \mathit{choice}_{k\in K_j}(b_{jk}\&(x_{jk} := e_{jk})\ Q_{jk})\}$$
$$[\!][\!]_{l\in L}\{@c_l\ R_l\}$$

Then

$$\begin{aligned} & P \parallel_r Q \\ = {} & [\!]_{j\in J}\{[q_j]\ \mathit{choice}_{k\in K_j}(b_{ij}\&(x_{jk} := e_{jk})\ \mathbf{par}(P_{jk}, Q, r))\} \\ & [\!][\!]_{i\in I}\{@(b_i \wedge \neg c)\ \mathbf{par}(P_i, Q, r)\} \\ & [\!][\!]_{l\in L}\{@(c_l \wedge \neg b)\ \mathbf{par}(P, R_l, r)\} \\ & [\!][\!]_{i\in I\wedge l\in L}\{@(b_i \wedge c_l)\ \mathbf{par}(P_i, Q_l, r)\} \end{aligned}$$

where, $b = \vee_{i\in I}\, b_i$ and $c = \vee_{l\in L}\, c_l$

We shall next consider the parallel composition where the first component is a time-delay guarded construct. We present one law (par-3-10) below with the other two listed in Appendix A.

The following law captures the case where the second component is also a time-delay guarded construct. The whole process performs a time delay and then behaves as the parallel composition of the remaining parts from both sides:

(par-3-10) Let $P = [\!]\{\#1\ R\}$ and $Q = [\!]\{\#1\ T\}\}$

Then

$$P \parallel_r Q = [\!]\{\#1\ \mathbf{par}(R, T, r)\}$$

We now move to the parallel composition where the first component comprises both assignment-guarded com-

ponents and event-guarded components. The following law (par-3-14) captures the scenario where the second component consists of both event-guarded choices and time-delays:

(par-3-14) Let

$$P = [\!]_{i\in I}\{[p_i]\, choice_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\, P_{ij})\} \; [\!][\!]_{k\in K}\{@b_k\, R_k\}$$

and $Q = [\!]_{l\in L}\{@c_l\, Q_l\}[\!]\{\#1\, T\}$

Then

$$P \parallel_r Q$$
$$= [\!]_{i\in I}\{[p_i]\, choice_{j\in J}(b_{ij}\&(x_{ij} := e_{ij})\, \mathbf{par}(P_{ij}, Q, r))\}$$
$$[\!][\!]_{k\in K}\{@(b_k \wedge \neg c)\, \mathbf{par}(R_k, Q, r)\}$$
$$[\!][\!]_{l\in L}\{@(c_l \wedge b)\, \mathbf{par}(P, Q_l, r)\}$$
$$[\!][\!]_{k\in K\wedge l\in L}\{@(b_k \wedge c_l)\, \mathbf{par}(R_k, Q_l, r)\}$$

where, $b = \vee_{k\in K}\, b_k$ and $c = \vee_{l\in L}\, c_l$

Another similar law (par-3-13) is left in Appendix A.

The following law (par-3-15) is about the parallel composition of two guarded choices composing of both event-guarded components and time delay components.

(par-3-15) Let $P = [\!]_{i\in I}\{@b_i\, P_i\}[\!]\{\#1\, R\}$ and $Q = [\!]_{j\in J}\{@c_j\, Q_j\}[\!]\{\#1\, T\}$

Then

$$P \parallel_r Q$$
$$= [\!]_{k\in K}\{@(b_i \wedge \neg c)\, \mathbf{par}(P_i, Q, r)\}$$
$$[\!][\!]_{j\in J}\{@(c_j \wedge \neg b)\, \mathbf{par}(P, Q_j, r)\}$$
$$[\!][\!]_{j\in J}\{@(b_i \wedge c_j)\, \mathbf{par}(P_i, Q_j, r)\}$$
$$[\!]\{\#1\, \mathbf{par}(R, T, r)\}$$

where, $b = \vee_{i\in I}\, b_i$ and $c = \vee_{j\in J}\, c_j$

6 Animation of Operational Semantics

Operational semantics provides a set of transition rules that models how a program performs step by step. If we can have an executed version of operational semantics, the correctness of operational semantics can be checked from various test results. This means that a simulator for the operational semantics of our proposed language is highly desirable.

Transition rules for the operational semantics can be translated into Prolog logic programming clauses [3]. Prolog has been successfully applied in rapid-prototyping, including [1, 2]. Building on this, for the development of the simulator of *PTSC*, we have selected Prolog as our programming language.

The configuration in a transition can be expressed as a list (indicated by square brackets) in Prolog:

$$[\, P,\ Sigma\,]$$

where state $Sigma$ can also be implemented as a list that contains values for program variables.

The transitions for the operational semantics can be directly translated into Prolog clauses. For some individual specific transitions, there may be several transitions for that kind of transition expressed in Prolog because it covers several different cases. For example, for the transition (3) of parallel composition (see page 4):

If $\langle P,\ \sigma\rangle \xrightarrow{c}_{p_2} \langle P',\ \sigma'\rangle$ and $stable(\langle x, \sigma\rangle)$ and $stableE(\langle x, \sigma\rangle)$ $(x = P,\ Q)$,

then $\langle P \parallel_{p_1} Q,\ \sigma\rangle \xrightarrow{c}_{p_1\times p_2} \langle \mathbf{par}(P', Q, p_1),\ \sigma'\rangle$.

$\langle Q \parallel_{p_1} P,\ \sigma\rangle \xrightarrow{c}_{(1-p_1)\times p_2} \langle \mathbf{par}(Q, P', p_1),\ \sigma'\rangle$.

This transition can be translated into Prolog clauses shown as below. Every clause can represent the specific individual case.

$$\frac{[\,S1,\ Sigma\,] -['c', Y] \longrightarrow [\,epsilon,\ Sigma1\,] \wedge stableB(S1, S2, Sigma)}{[\,S1|X|S2,\ Sigma\,] -['c', X * Y] \longrightarrow [\,S2,\ Sigma1\,]}$$

$$\frac{[\,S1,\ Sigma\,] -['c', Y] \longrightarrow [\,epsilon,\ Sigma1\,] \wedge stableB(S1, S2, Sigma)}{[\,S2|X|S1,\ Sigma\,] -['c', (1 - X) * Y] \longrightarrow [\,S2,\ Sigma1\,]}$$

$$\frac{[\,S1,\ Sigma\,] -['c', Y] \longrightarrow [\,S11,\ Sigma1\,] \wedge S11 \sim= epsilon \wedge stableB(S1, S2, Sigma)}{[\,S1|X|S2,\ Sigma\,] -['c', X * Y] \longrightarrow [\,S11|X|S2,\ Sigma1\,]}$$

$$\frac{[\,S1,\ Sigma\,] -['c', Y] \longrightarrow [\,S11,\ Sigma1\,] \wedge S11 \sim= epsilon \wedge stableB(S1, S2, Sigma)}{[\,S2|X|S1,\ Sigma\,] -['c', (1 - X) * Y] \longrightarrow [\,S2|X|S11,\ Sigma1\,]}$$

where,

$$stableB(P, Q, \sigma) =_{df} stable(\langle P, \sigma\rangle) \wedge stableE(\langle P, \sigma\rangle) \wedge stable(\langle Q, \sigma\rangle) \wedge stableE(\langle Q, \sigma\rangle).$$

Here, we use "$S1\,|X|\,S2$" to represent "$S1 \parallel_X S2$" and "$-['c', X] \longrightarrow$" to represent "$\xrightarrow{c}_X$" in the Prolog clauses. Meanwhile, "$epsilon$" stands for the empty process ε and $\sim=$ stands for $\neq$.

Next we use the example below to demonstrate how the simulator works. Consider the program $(x := x + 1\ ;\ x := x + 2) \parallel_{0.4} (x := 2\ ;\ x := 4)$. From the execution based on the simulator, we know there are six execution sequences leading the program to the terminating state, where the transitions contain their own specific probability [1].

```
? - track [(x := x + 1; x := x + 2) |0.4| (x := 2; x := 4), [x = 0]].   (1)
1 -[0, 0.4]-> [x = x + 2 |0.4| x = 2; x = 4, [x = 1]]                   (2)
2 -[0, 0.4]-> [x = 2; x = 4, [x = 3]]                                   (3)
3 -[0, 1]-> [x = 4, [x = 2]]                                            (4)
4 -[0, 1]-> [epsilon, [x = 4]]                                          (5)
2 -[0, 0.6]-> [x = x + 2 |0.4| x = 4, [x = 2]]                          (6)
3 -[0, 0.4]-> [x = 4, [x = 4]]                                          (7)
4 -[0, 1]-> [epsilon, [x = 4]]                                          (8)
3 -[0, 0.6]-> [x = x + 2, [x = 4]]                                      (9)
4 -[0, 1]-> [epsilon, [x = 6]]                                          (10)
1 -[0, 0.6]-> [x = x + 1; x = x + 2 |0.4| x = 4, [x = 2]]               (11)
2 -[0, 0.6]-> [x = x + 1; x = x + 2, [x = 4]]                           (12)
3 -[0, 1]-> [x = x + 2, [x = 5]]                                        (13)
4 -[0, 1]-> [epsilon, [x = 7]]                                          (14)
2 -[0, 0.4]-> [x = x + 2 |0.4| x = 4, [x = 3]]                          (15)
3 -[0, 0.4]-> [x = 4, [x = 5]]                                          (16)
4 -[0, 1]-> [epsilon, [x = 4]]                                          (17)
3 -[0, 0.6]-> [x = x + 2, [x = 4]]                                      (18)
4 -[0, 1]-> [epsilon, [x = 6]]                                          (19)
```

[1] The notation $[X, Y]$ in the displayed execution sequence indicates that X is the duration of the transition and Y is the probability of the transition. Meanwhile, "$track$" is the command in the simulator to display the execution sequence.

In total, there are six execution sequences leading the original program to the terminating state, where each transition step contains probability either 0.4, 0.6 or 1. These six execution sequences are:

$(1)(2)(3)(4)(5)$ $(1)(2)(6)(7)(8)$
$(1)(2)(6)(9)(10)$ $(1)(11)(12)(13)(14)$
$(1)(11)(15)(16)(17)$ $(1)(11)(15)(18)(19)$

where (i) stands for row i (given at the end of row i).

In summary, the simulator can display the execution sequences of programs based on the operational semantics. It provides an animation tool for our operational semantics. From various test examples including those above, the results displayed give us additional confidence concerning the validity of the operational semantics.

7 Conclusion

In this paper we integrated probability with a timed concurrent language. The probability was added into non-deterministic choice, parallel composition and guarded choice. Parallel processes communicate with each other via shared variables. We formalized a structural operational semantics for the language which incorporates time, concurrency and probability. On top of the operational model, an abstract bisimulation relation was defined and a rich set of algebraic laws have been derived for program equivalence. A prototype was also built for the animation of the execution of probabilistic programs. As an immediate future work, it would be interesting to work out a denotational model and investigate the consistency between operational and denotational models. We would like to explore an observation-oriented model as advocated in *UTP* [12], which, we believe, would make the linking theory easier to build.

References

[1] J. P. Bowen. Combining operational semantics, logic programming and literate programming in the specification and animation of the Verilog Hardware Description Language. In *Proc. IFM 2000: 2nd International Conference on Integrated Formal Methods*, volume 1945 of *Lecture Notes in Computer Science*, pages 277–296. Springer-Verlag, November 2000.

[2] J. P. Bowen, He Jifeng, and Xu Qiwen. An animatable operational semantics of the Verilog Hardware Description Language. In *Proc. ICFEM 2000: 3rd IEEE International Conference on Formal Engineering Methods*, pages 199–207. IEEE Computer Society Press, September 2000.

[3] W. F. Clocksin and C. S. Mellish. *Programming in Prolog*. Springer-Verlag, 5th edition, 2003.

[4] J. de Bakker and E. de Vink. *Control Flow Semantics*. The MIT Press, 1996.

[5] J. den Hartog. *Probabilistic Extensions of Semantic Models*. PhD thesis, Vrije University, The Netherlands, 2002.

[6] J. den Hartog and E. de Vink. Mixing up nondeteminism and probability: A premliminary report. In *Proc PROBMIV'98: Workshop on Probabilistic Methods in Verification,* June, 1998, volume 22 of Electronic Notes in Theoretical Computer Science. Elsevier, 1999.

[7] J. den Hartog and E. de Vink. Verifying probabilistic programs using a Hoare like logic. *International Journal of Foundations of Computer Science*, 40(3):315–340, 2002.

[8] J. den Hartog, E. de Vink, and J. de Bakker. Metrix semantics and full abstractness for action refinement and probabilistic choice. In *Proc MFCSIT: 1st Irish Conference on the Mathematical Foundation of Computer Science and Information Technology,* July, 2000, volume 40 of Electronic Notes in Theoretical Computer Science. Elsevier, 2001.

[9] He Jifeng. *Provably Correct Systems: Modelling of Communication Languages and Design of Optimized Compilers*. The McGraw-Hill International Series in Software Engineering, 1994.

[10] He Jifeng, K. Seidel, and A. McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28(2-3):171–192, 1997.

[11] He Jifeng and Zhu Huibiao. Formalising Verilog. In *Proc. ICECS 2000: IEEE International Conference on Electronics, Circuits and Systems*, pages 412–415. IEEE Computer Society Press, December 2000.

[12] C. A. R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice Hall International Series in Computer Science, 1998.

[13] A. McIver and C. Morgan. Partial correctness for probabilistic demonic programs. *Theoretical Computer Science*, 266(1-2):513–541, 2001.

[14] A. McIver and C. Morgan. *Abstraction, Refinement and Proof of Probability Systems*. Monographs in Computer Science. Springer, October 2004.

[15] A. McIver, C. Morgan, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, 1996.

[16] R. Milner. *Communication and Concurrency*. Prentice Hall International Series in Computer Science, 1990.

[17] R. Milner. *Communication and Mobile System: π-calculus*. Cambridge University Press, 1999.

[18] M. Núñez. Algebraic theory of probabilistic processes. *The Journal of Logic and Algebraic Programming*, 56:117–177, 2003.

[19] M. Núñez and D. de Frutos-Escrig. Testing semantics for probabilistic LOTOS. In *Proc FORTE'95: IFIP TC6 Eighth International Conference on Formal Description Techniques,* Montreal, Canada, October 1995, volume 43 of *IFIP Conference Proceedings*, pages 367–382. Chapman & Hall, 1996.

[20] M. Núñez, D. de Frutos-Escrig, and L. F. L. Díaz. Acceptance trees for probabilistic processes. In *Proc CONCUR'95: 6th International Conference on Concurrency,* Philadelphia, PA, USA, August, 1995, volume 962 of *Lecture Notes in Computer Science*. Springer, 1995.

[21] S. Park, F. Pfenning, and S. Thrun. A probabilistic language based upon sampling functions. In *Proc POPL 2005: 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 171–182. ACM, January 2005.

[22] G. D. Plotkin. A structural approach to operational semantics. Technical Report 19, University of Aarhus, Denmark, 1981. Also published in *The Journal of Logic and Algebraic Programming*, volumes 60/61:17–139, 2004.

[23] Zhu Huibiao. *Linking the Semantics of a Multithreaded Discrete Event Simulation Language*. PhD thesis, London South Bank University, UK, February 2005.

[24] Zhu Huibiao and He Jifeng. A semantics of Verilog using Duration Calculus. In *Proc. International Conference on Software: Theory and Practice*, pages 421–432, August 2000.

Appendix

A. More Parallel Expansion Laws

(par-3-4) Let

$P = \|_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$ and

$Q = \|_{k\in K}\{[q_k]\ \mathit{choice}_{l\in L_k}(b_{kl}\&(x_{kl} := e_{kl})\ Q_{kl})\}$
$\|\|_{m\in M}\{@c_m\ R_m\}$

Then

$$\begin{aligned} & P \parallel_r Q \\ = \ & \|_{i\in I}\{[r\times p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij}) \\ & \qquad\qquad \mathbf{par}(P_{ij}, Q, r))\} \\ & \|\|_{k\in K}\{[(1-r)\times q_k]\ \mathit{choice}_{l\in L_k}(b_{kl}\&(x_{kl} := e_{kl}) \\ & \qquad\qquad \mathbf{par}(P, Q_{kl}, r)\} \\ & \|\|_{m\in M}\{@c_k\ \mathbf{par}(P, R_m, r)\} \end{aligned}$$

(par-3-5) Let

$P = \|_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$ and

$Q = \|_{l\in L}\{@c_l\ Q_l\}\|\{\#1\ R\}$

Then

$$\begin{aligned} & P \parallel_r Q \\ = \ & \|_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ \mathbf{par}(P_{ij}, Q, r))\} \\ & \|\|_{l\in L}\{@c_l\ \mathbf{par}(P, Q_l, r)\} \end{aligned}$$

(par-3-9) Let $P = \|_{i\in I}\{@b_i\ P_i\}$ and
$Q = \|_{j\in J}\{@c_j\ Q_j\}\|\{\#1\ R\}$

Then

$$\begin{aligned} & P \parallel_r Q \\ = \ & \|_{i\in I}\{@(b_i \wedge \neg c)\ \mathbf{par}(P_i, Q, r)\} \\ & \|\|_{j\in J}\{@(c_j \wedge \neg b)\ \mathbf{par}(P, Q_j, r)\} \\ & \|\|_{i\in I\wedge j\in J}\{@(b_i \wedge c_j)\ \mathbf{par}(P_i, Q_j, r)\} \\ & \|\{\#1\ \mathbf{par}(P, R, r)\} \end{aligned}$$

where, $b = \vee_{i\in I}\, b_i$ and $c = \vee_{j\in J}\, c_j$

(par-3-11) Let

$P = \|\{\#1\ T\}$ and

$Q = \|_{i\in I}\{[q_i]\ \mathit{choice}_{j\in J_i}(b_{jk}\&(x_{ij} := e_{ij})\ Q_{ij})\}$
$\|\|_{k\in K}\{@c_k\ R_k\}$

Then

$$\begin{aligned} & P \parallel_r Q \\ = \ & \|_{i\in I}\{[q_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ \mathbf{par}(P, Q_{ij}, r))\} \\ & \|\|_{k\in K}\{@c_k\ \mathbf{par}(P, R_k, r)\} \end{aligned}$$

(par-3-12) Let $P = \|\{\#1\ T\}$ and
$Q = \|_{i\in I}\{@b_i\ Q_i\}\|\{\#1\ R\}$

Then

$P \parallel_r Q = \|_{i\in I}\{@b_i\ \mathbf{par}(P, Q_i, r)\}\|\{\#1\ \mathbf{par}(T, R, r)\}$

(par-3-13) Let

$P = \|_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$
$\|\|_{k\in K}\{@b_k\ R_k\}$

and

$Q = \|_{l\in L}\{[q_l]\ \mathit{choice}_{m\in M_l}(c_{lm}\&(x_{lm} := e_{lm})\ P_{lm})\}$
$\|\|_{n\in N}\{@c_n\ T_n\}$

Then

$$\begin{aligned} & P \parallel_r Q \\ = \ & \|_{i\in I}\{[r\times p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ \mathbf{par}(P_{ij}, Q, r))\} \\ & \|\|_{l\in L}\{[(1-r)\times q_l]\ \mathit{choice}_{m\in M_l}(c_{lm}\&(x_{lm} := e_{lm}) \\ & \qquad\qquad \mathbf{par}(P, Q_{lm}, r))\} \\ & \|\|_{k\in K}\{@(b_k \wedge \neg c)\ \mathbf{par}(R_k, Q, r)\} \\ & \|\|_{n\in N}\{@(c_n \wedge \neg b)\ \mathbf{par}(R_k, Q, r)\} \\ & \|\|_{k\in K\wedge n\in N}\{@(b_k \wedge c_n)\ \mathbf{par}(R_k, Q_n, r)\} \end{aligned}$$

where, $b = \vee_{k\in K}\, b_k$ and $c = \vee_{n\in N}\, c_n$

B. Proof of Theorem 3: $\approx$ is a congruence.

Assume $P \approx Q$. We know for any state σ, $\langle P, \sigma\rangle \approx \langle Q, \sigma\rangle$. This means there exists a bisimulation S_σ such that $\langle P, \sigma\rangle \approx \langle Q, \sigma\rangle$. Let $S = \cup_\sigma\, S_\sigma$. We know S is also a bisimulation.

(1) For the proof of $P\ ;\ R \approx Q\ ;\ R$, let

$S_{1,1} =_{df} Id \cup \{\ (\langle P; R,\ \sigma\rangle,\ \langle Q; R,\ \sigma\rangle)\ \mid\ \langle P,\ \sigma\rangle\ S\ \langle Q,\ \sigma\rangle\ \}$

For the proof of $R\ ;\ P \approx R\ ;\ Q$, let

$S_{1,2} =_{df} S \cup \{\ (\langle R; P,\ \sigma\rangle,\ \langle R; Q,\ \sigma\rangle)\ \mid\ \langle P,\ \sigma\rangle\ S\ \langle Q,\ \sigma\rangle\ \}$

(2) For the proof of

$\mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ R \approx \mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ R,$

let

$$\begin{aligned} S_{2,1} =_{df}\ & Id \cup S \cup \\ & \{\ (\langle \mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ R,\ \sigma\rangle,\ \langle \mathbf{if}\ b\ \mathbf{then}\ Q\ \mathbf{else}\ R,\ \sigma\rangle \\ & \qquad\qquad \mid\ \langle P,\ \sigma\rangle\ S\ \langle Q,\ \sigma\rangle\ \}, \end{aligned}$$

For the proof of

$\mathbf{if}\ b\ \mathbf{then}\ R\ \mathbf{else}\ P \approx \mathbf{if}\ b\ \mathbf{then}\ R\ \mathbf{else}\ Q,$

let

$S_{2,2} =_{df} Id \cup S \cup$
$\{ (\langle \textbf{if } b \textbf{ then } R \textbf{ else } P, \sigma\rangle, \langle \textbf{if } b \textbf{ then } R \textbf{ else } Q, \sigma\rangle) \mid \langle P, \sigma\rangle S \langle Q, \sigma\rangle \}$

(3) For the proof of $\textbf{while } b \textbf{ do } P \approx \textbf{while } b \textbf{ do } Q$, let

S_3
$=_{df} Id \cup$
$\{ (\langle \textbf{while } b \textbf{ do } P, \sigma\rangle, \langle \textbf{while } b \textbf{ do } Q, \sigma\rangle) \mid \langle P, \sigma\rangle S \langle Q, \sigma\rangle \} \cup$
$\{ (\langle U; \textbf{while } b \textbf{ do } P, \sigma\rangle, \langle V; \textbf{while } b \textbf{ do } Q, \sigma\rangle) \mid \langle P, \sigma\rangle S \langle Q, \sigma\rangle \wedge \langle U, \sigma\rangle S \langle V, \sigma\rangle \}$

(4) For the proof of $P \sqcap R \approx Q \sqcap R$, let

$S_{4,1} =_{df} Id \cup S \cup$
$\{(\langle P \sqcap R, \sigma\rangle, \langle Q \sqcap R, \sigma\rangle) \mid \langle P, \sigma\rangle S \langle Q, \sigma\rangle\}$

For the proof of $R \sqcap P \approx R \sqcap Q$, let

$S_{4,2} =_{df} Id \cup S \cup$
$\{(\langle R \sqcap P, \sigma\rangle, \langle R \sqcap Q, \sigma\rangle) \mid \langle P, \sigma\rangle S \langle Q, \sigma\rangle\}$

(5) For the proof of $P \sqcap_p R \approx Q \sqcap_p R$, let

$S_{5,1} =_{df} Id \cup S \cup$
$\{(\langle P \sqcap_p R, \sigma\rangle, \langle Q \sqcap_p R, \sigma\rangle) \mid \langle P, \sigma\rangle S \langle Q, \sigma\rangle\}$

For the proof of $R \sqcap_p P \approx R \sqcap_p Q$, let

$S_{5,2} =_{df} Id \cup S \cup$
$\{(\langle R \sqcap_p P, \sigma\rangle, \langle R \sqcap_p Q, \sigma\rangle) \mid \langle P, \sigma\rangle S \langle Q, \sigma\rangle\}$

(6) For the proof of the probabilistic guarded choice, without loss of generality, we only consider the first type of guarded choice here. Let

$T_1 = [\![\{[p]\, choice(b\&(x := e)\, P,\ G1),\ G2\}$ and
$T_2 = [\![\{[p]\, choice(b\&(x := e)\, Q,\ G1),\ G2\}$

In order to consider the proof of $T_1 \approx T_2$, let

S_6
$=_{df} Id \cup S \cup$
$\{(\langle T_1, \sigma\rangle, \langle T_2, \sigma\rangle) \mid \langle P, \sigma\rangle, S \langle Q, \sigma\rangle\}$

(7) For the proof of $P \parallel_p R \approx Q \parallel_p R$, let

$S_{7,1}$
$=_{df} Id \cup S \cup$
$\{(\langle P \parallel_p R, \sigma\rangle, \langle Q \parallel_p R, \sigma\rangle) \mid \langle P, \sigma\rangle S \langle Q, \sigma\rangle\}$

For the proof of $R \parallel_p P \approx R \parallel_p Q$, let

$S_{7,2}$
$=_{df} Id \cup S \cup$
$\{(\langle R \parallel_p P, \sigma\rangle, \langle R \parallel_p P, \sigma\rangle) \mid \langle P, \sigma\rangle S \langle Q, \sigma\rangle\}$

We can show that each $S_{i,j}$ (or S_i)is a bisimulation. □

C. Proof of Law (*prob-3)*

Now we give the proof for the algebraic law (prob-3) (see page 6).

Let

$S =_{df} \{ (\langle P \sqcap_{p_1} (Q \sqcap_{p_2} R), \sigma\rangle, \langle (P \sqcap_x Q) \sqcap_y R), \sigma\rangle) \mid P, Q, R \text{ are programs } \wedge \sigma \in \Sigma \wedge p1, p2 \in (0, 1) \}$

where, (1) $x = p_1/(p_1 + p_2 - p_1 \times p_2)$,
$y = p_1 + p_2 - p_1 \times p_2$
(2) Σ denaotes the set containing all the states.

Further, let $T =_{df} Id \cup S \cup S^{-1}$.

Now we need to prove that T is a bisimulation relation.

(1) From the transitions of $\sqcap_p$, we know that both $\langle P \sqcap_{p_1} (Q \sqcap_{p_2} R), \sigma\rangle$ and $\langle (P \sqcap_x Q) \sqcap_y R), \sigma\rangle$ cannot do transition of type $\xrightarrow{\tau}$ and $\xrightarrow{v}$. This indicates that we don't need to check the first item of bisimulation definition.

(2) Now we need to prove the item (2) of bisimulation relation for T.

(a) If $\langle P, \sigma\rangle \stackrel{c}{\Longrightarrow}_{n'_1,p'_1} \langle P', \sigma'\rangle$,
then $\langle P \sqcap_{p_1} (Q \sqcap_{p_2} R), \sigma\rangle \stackrel{c}{\Longrightarrow}_{n'_1,p_1\times p'_1} \langle P', \sigma'\rangle$
and
$\langle (P \sqcap_x Q) \sqcap_y R, \sigma\rangle \stackrel{c}{\Longrightarrow}_{n'_1,u_1} \langle P', \sigma'\rangle$
where, $u_1 = y \times x \times p'_1$.
From x and y, we know $u_1 = p_1 \times p'_1$.

(b) If $\langle Q, \sigma\rangle \stackrel{c}{\Longrightarrow}_{n'_2,p'_2} \langle Q', \sigma'\rangle$,
then $\langle P \sqcap_{p_1} (Q \sqcap_{p_2} R), \sigma\rangle \stackrel{c}{\Longrightarrow}_{n'_2,(1-p_1)\times p_2\times p'_2} \langle P', \sigma'\rangle$
and
$\langle (P \sqcap_x Q) \sqcap_y R, \sigma\rangle \stackrel{c}{\Longrightarrow}_{n'_2,u_2} \langle P', \sigma'\rangle$
where, $u_2 = y \times (1 - x) \times p'_2$.
From x and y, we know that $u_2 = (1 - p_1) \times p_2 \times p'_2$

(c) If $\langle R, \sigma\rangle \stackrel{c}{\Longrightarrow}_{n'_3,p'_3} \langle R', \sigma'\rangle$,
then
$\langle P \sqcap_{p_1} (Q \sqcap_{p_2} R), \sigma\rangle \stackrel{c}{\Longrightarrow}_{n'_1,(1-p_1)\times(1-p_2)\times p'_3} \langle P', \sigma'\rangle$
and
$\langle (P \sqcap_x Q) \sqcap_y R, \sigma\rangle \stackrel{c}{\Longrightarrow}_{n'_3,u_3} \langle P', \sigma'\rangle$
where, $u_3 = (1 - y) \times p'_3$.
From x and y, we know $u_3 = (1-p_1)\times(1-p_2)\times p'_3$.

The above analysis leads to the satisfactory of the item (2) of bisimulation definition for the pair of configurations $(\langle P \sqcap_{p_1} (Q \sqcap_{p_2} R), \sigma\rangle, \langle (P \sqcap_x Q) \sqcap_y R), \sigma\rangle)$.

(3) The proof of item (3) of bisimulation relation for T is similar to the above proof of item (2). □